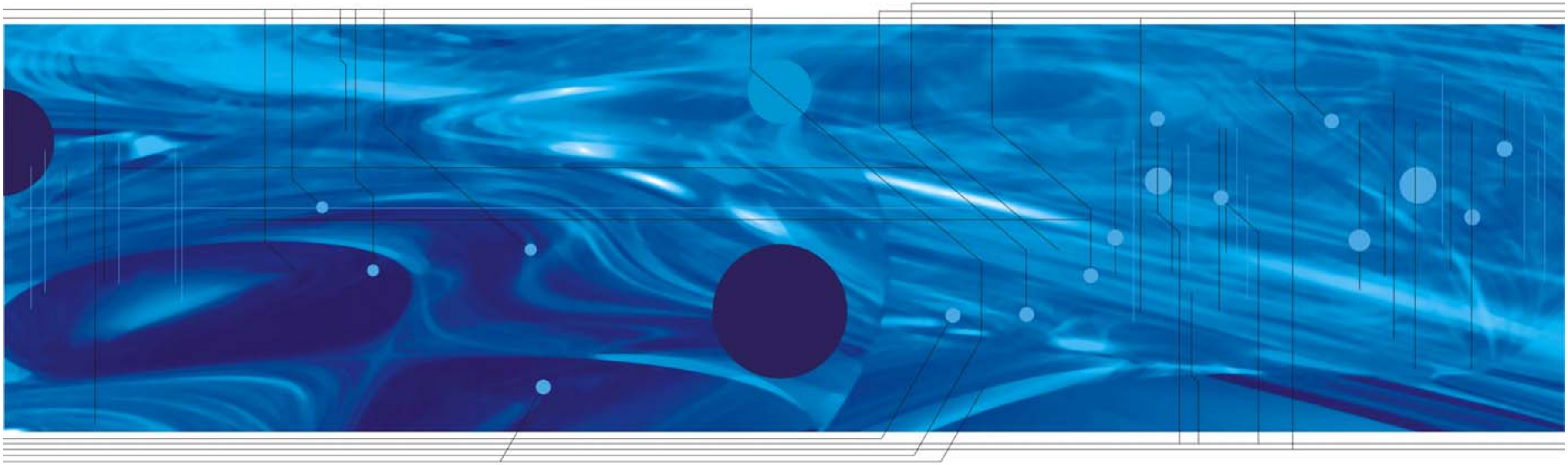


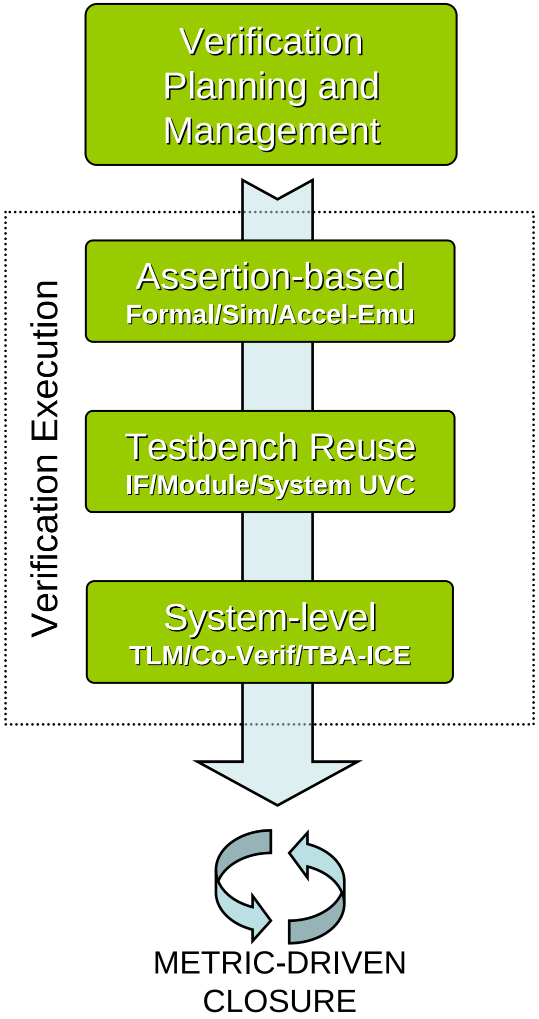
Plan to Silicon

Functional Test Automation using the Incisive platform and Plan to Closure Methodology

Session 1.15
CDNLive! Silicon Valley
September 2007

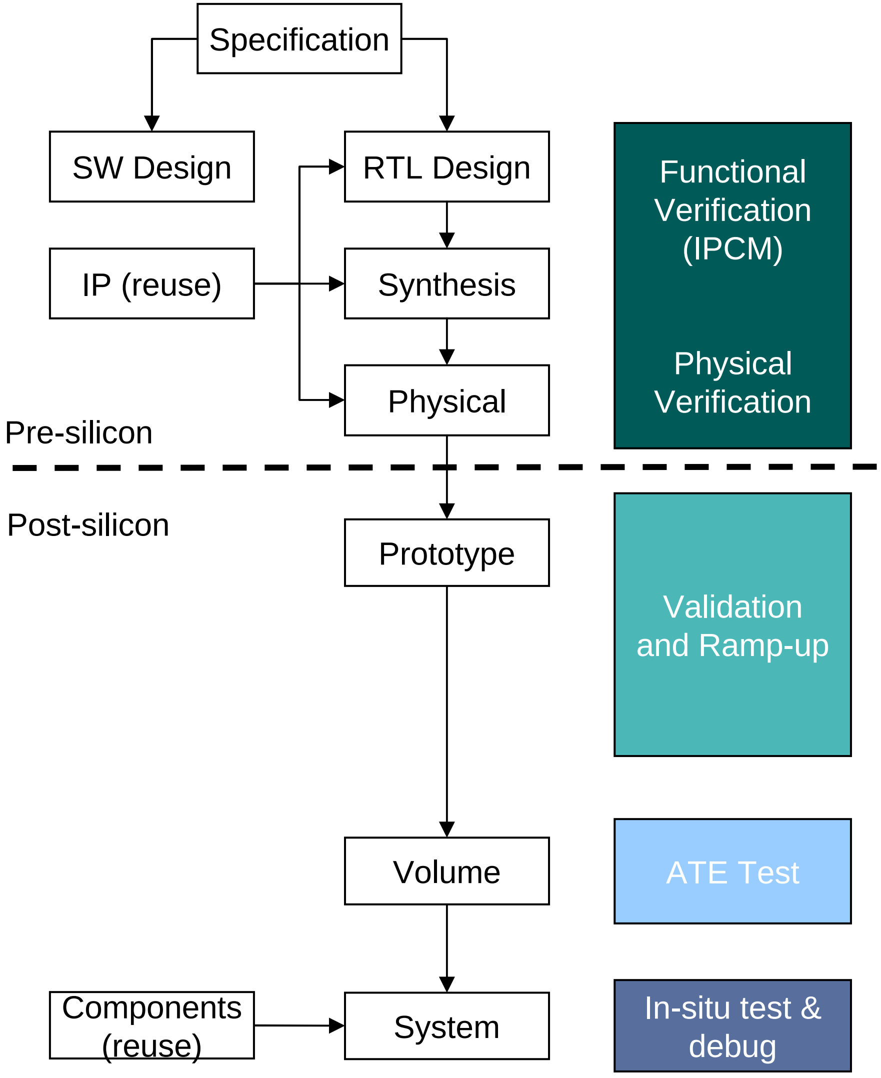
IPCM Overview

- Industry’s most complete and reliable flow from design specification to low-risk design closure
- Addresses:
  - Verification planning and management
  - Assertion-based verification
  - Testbench automation and reuse
  - Full system verification
- Process-driven flow based on metrics
- Highly-reusable and IP-ready



Ref: www.cadence.com

Validation Overview



Specification

SW Design

RTL Design

IP (reuse)

Synthesis

Physical

Pre-silicon

Post-silicon

Prototype

Volume

Components (reuse)

System

Functional Verification (IPCM)

Physical Verification

Validation and Ramp-up

ATE Test

In-situ test & debug

- Verify logic functionality
- Identify design bugs
- Ensure standard compliance and interoperability
- Complete closure targets

- Identify logic design bugs
- Identify operating corners
- Identify fab defects
- Signal integrity
- Power and thermal

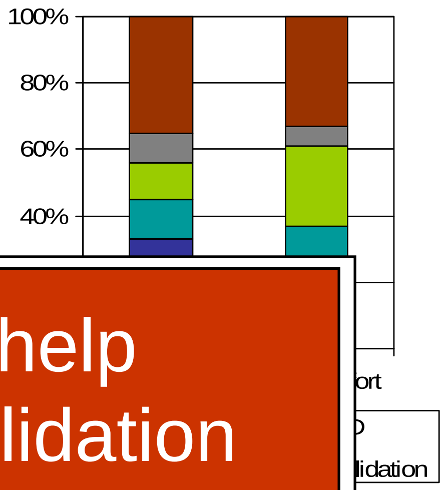
- Identify fab defects
- Resolve yield loss fast

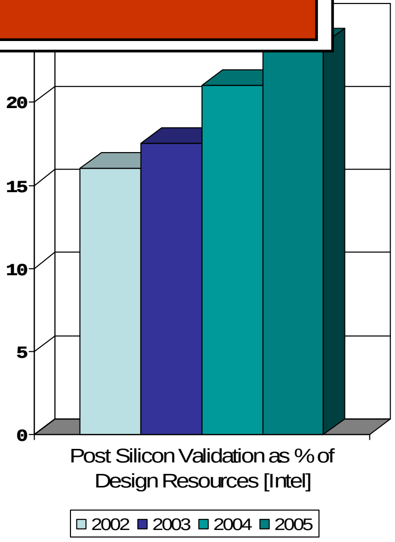
- Identify interoperabiltity limiting conditions
- Resolve yield lost very fast

Silicon Validation challenge

- Post-silicon validation cost is rising faster than design cost [Intel, ITC'06]
  - Design does not account for all parameters im
- Ramp
  - 25
- At 65 affect
  - Silicon debug is time-critical
  - >10% yield loss for 6 wks of production can be >$100M
- At system, no-trouble-found yield loss thins profit margins
  - Requirement to debug in-situ and analyze large data

Can IPCM be extended to help address the post-silicon validation challenge?



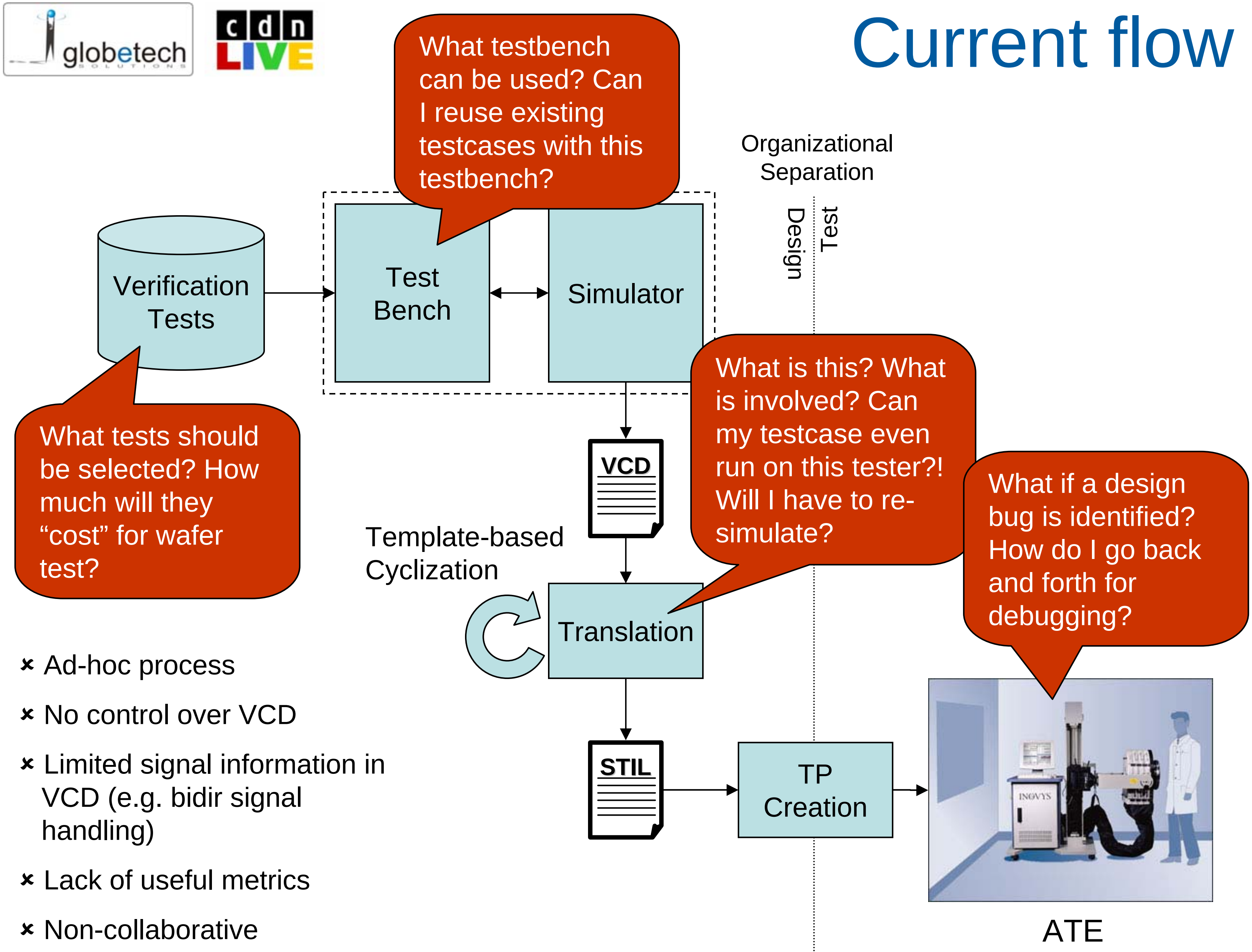


From Verification to Test

- Extending verification methodology can help address the post-silicon validation challenge
  - Leverage designer knowledge and experience
  - Reuse environments and tools
  - Know when to look when problems are identified
  - Build hardware support to enhance time-to-debug
  - Tackle massive unpredictability for TTx
- Looking at verification, post-silicon validation and silicon test from a methodology stand-point:
  - All domains need some form of test generation
  - All domains need arch/micro micro-arch knowledge
  - All domains need coverage metrics
  - All domains need some form of failure debug
  - Design hooks for one domain can benefit another

Ref: Seva Yerramili, Intel, ITC 2006

Current flow



Verification Tests

Test Bench

Simulator

VCD

Template-based Cyclization

Translation

STIL

TP Creation

ATE

Organizational Separation

Design

Test

What testbench can be used? Can I reuse existing testcases with this testbench?

What tests should be selected? How much will they “cost” for wafer test?

What is this? What is involved? Can my testcase even run on this tester?! Will I have to re-simulate?

What if a design bug is identified? How do I go back and forth for debugging?

- Ad-hoc process
- No control over VCD
- Limited signal information in VCD (e.g. bidir signal handling)
- Lack of useful metrics
- Non-collaborative organization separation

5 Distinct Problems

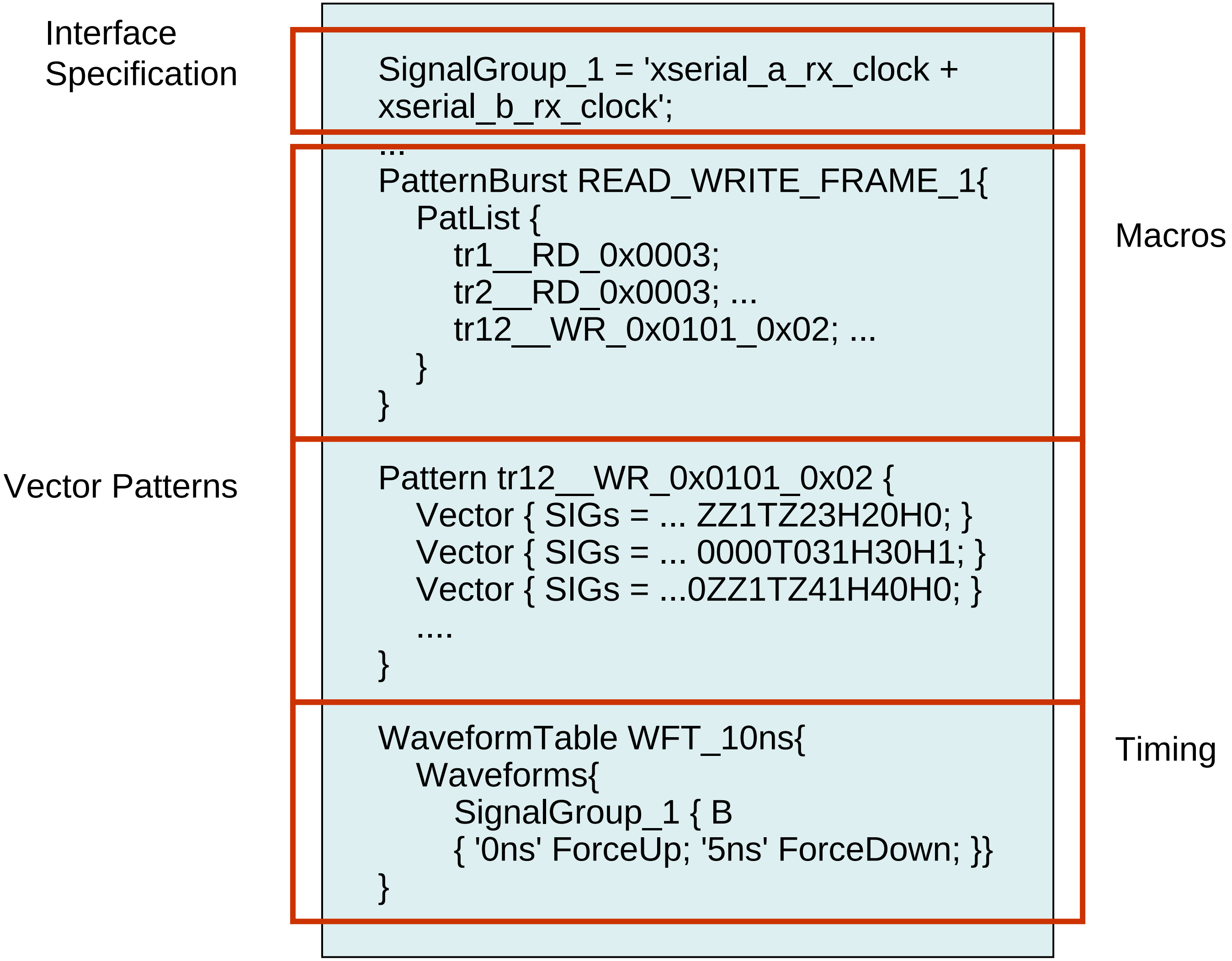
- How do I select appropriate tests
- How do I reuse testbenches
- How do I create working test programs
- How do I debug test programs and validate fixes
- How do I manage the validation process

Selecting Appropriate Tests

- Appropriate for what?
  - Functionality
    - All round IC exercise
    - Interface specific (DDR, PCIe, etc)
  - Design/Verification
    - Problem areas where many bugs were found
    - Areas were no bugs were found
    - Last minute design netlist changes
    - Functional coverage matrix (graded)
  - Physical
    - Maximum power draw
    - Minimum power draw (standby/hibernate)
    - Thermal or EMI
  - Many others….
- Keeping in mind
  - ATE imposes many constraints on test programs

Looking at a test program

IEEE 1450.1 Standard Test Interface Language (STIL)



Interface Specification

Macros

Vector Patterns

Timing

```
SignalGroup_1 = 'xserial_a_rx_clock +
xserial_b_rx_clock';
...
PatternBurst READ_WRITE_FRAME_1{
    PatList {
        tr1__RD_0x0003;
        tr2__RD_0x0003; ...
        tr12__WR_0x0101_0x02; ...
    }
}

Pattern tr12__WR_0x0101_0x02 {
    Vector { SIGs = ... ZZ1TZ23H20H0; }
    Vector { SIGs = ... 0000T031H30H1; }
    Vector { SIGs = ...0ZZ1TZ41H40H0; }
    ....
}

WaveformTable WFT_10ns{
    Waveforms{
        SignalGroup_1 { B
        { '0ns' ForceUp; '5ns' ForceDown; }}
}
```

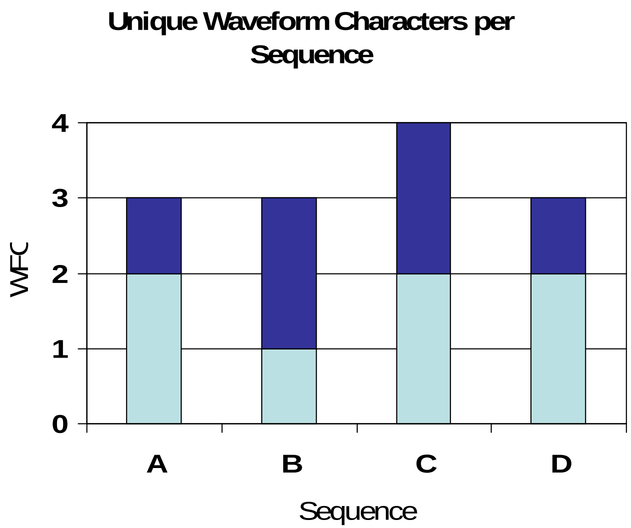
Functional test metrics

- Interfaces
  - Number of I/Os
  - Types of I/Os (bidir, high-speed, _oe, etc)
- Data
  - Number of vectors in set
  - Vector depth
- Timing
  - Cycle duration
  - Number of waveform tables
  - Number of waveforms
  - Number of waveform characters per waveform
- Semantics
  - Importance of vector sets (test coverage, functional coverage, etc)
  - Association of data sets with verification history (e.g. failing assertions)
  - Transaction grouping
  - Other…

Analyzing test programs

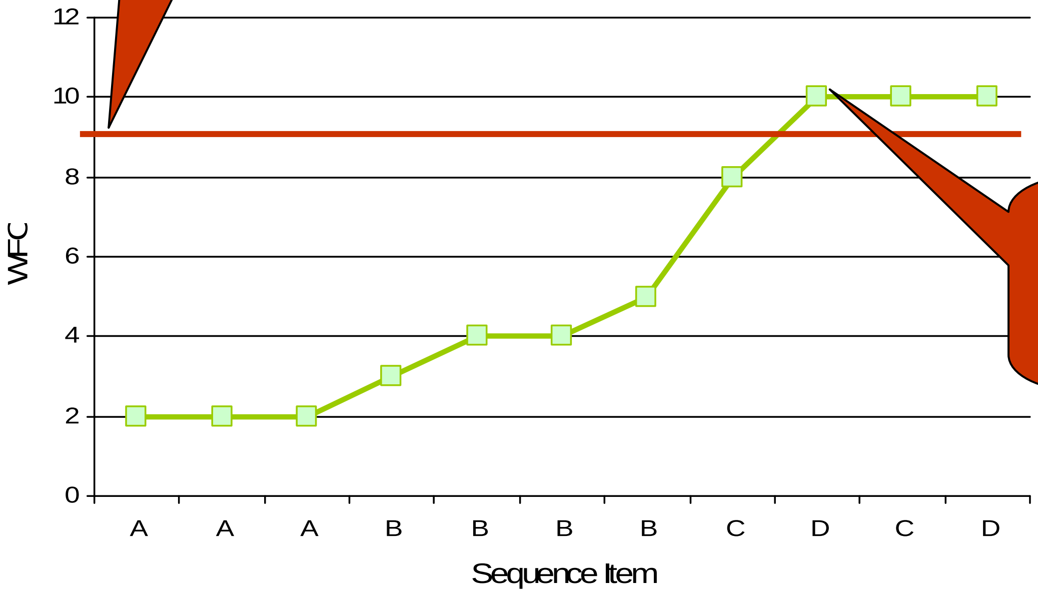
- Example verification scenario:
  - – Virtual sequence
    { A, A, A, B, B, B, B, C, D, C, D }

Unique Waveform Characters per Sequence



ATE Constraint: Waveform table limit == 9

Waveform Characters for Virtual Sequence

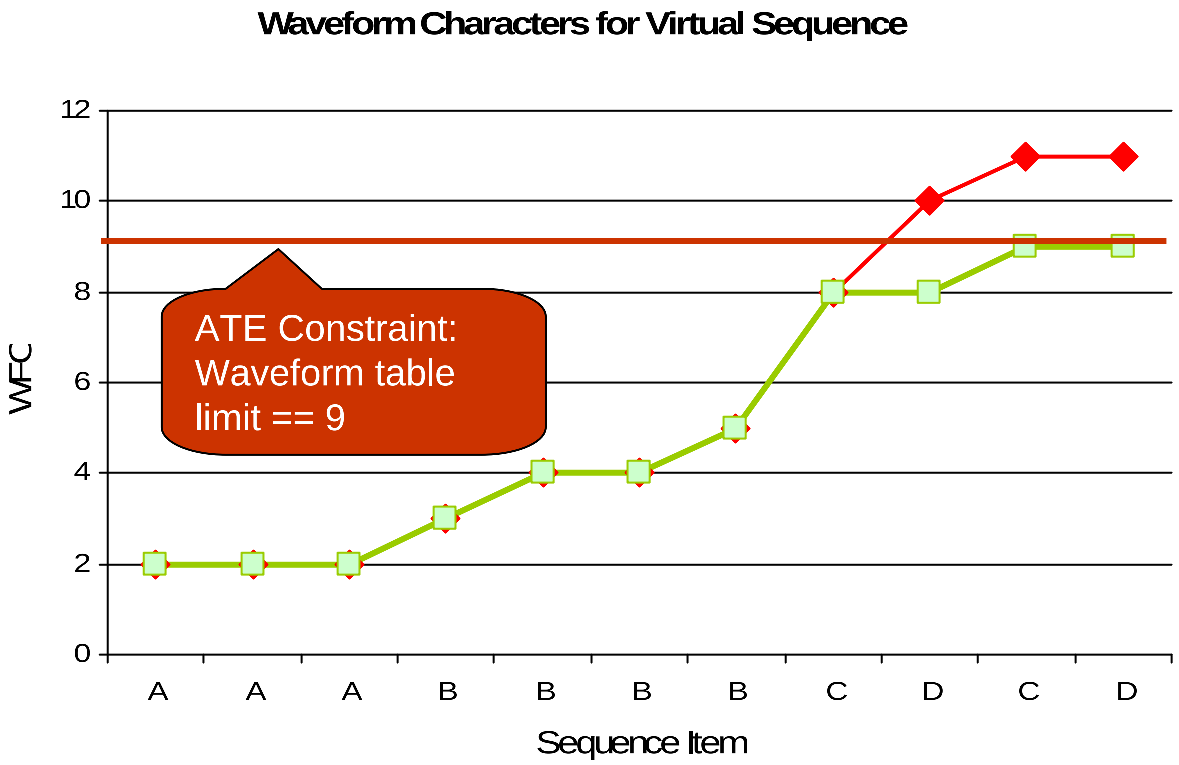


Test scenario violates ATE resource constraint for virtual sequence

Fix by Constraint

- Control WFC generation by constraint, e.g.:

```
extend C item_s {
  keep range in [0..2];
};
```

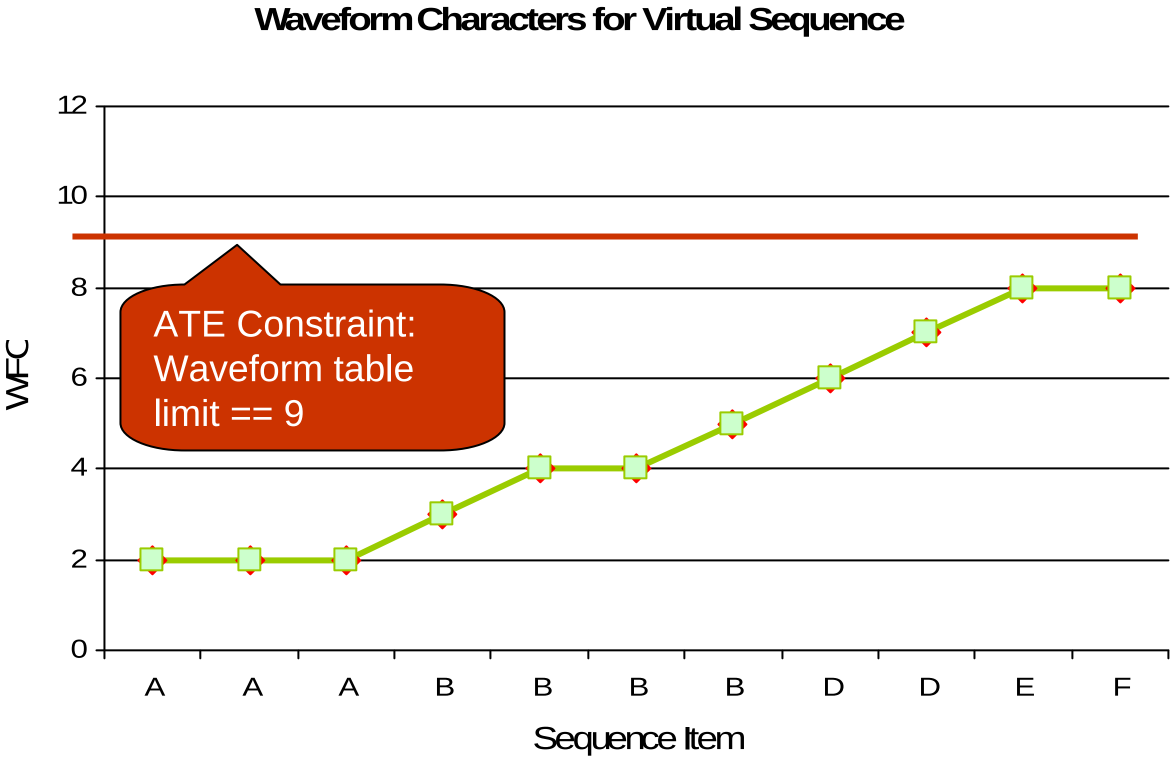


ATE Constraint: Waveform table limit == 9

Fix by Test Reordering

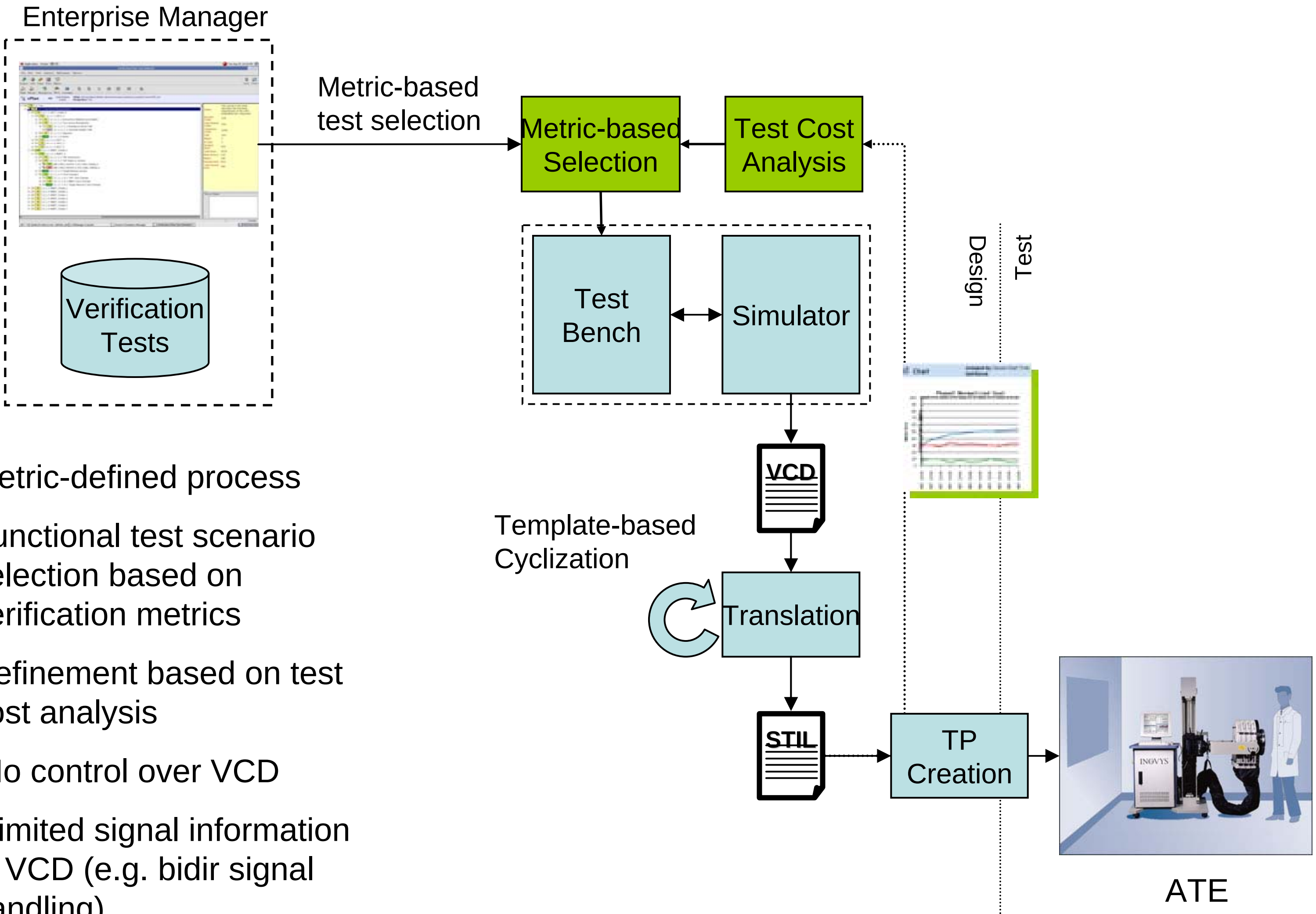
- After reviewing, decide to skip sequence ‘C’ altogether:

```
extend MAIN seq_s{
  do A seq;
  do B seq;
  do D seq;
  …
};
```



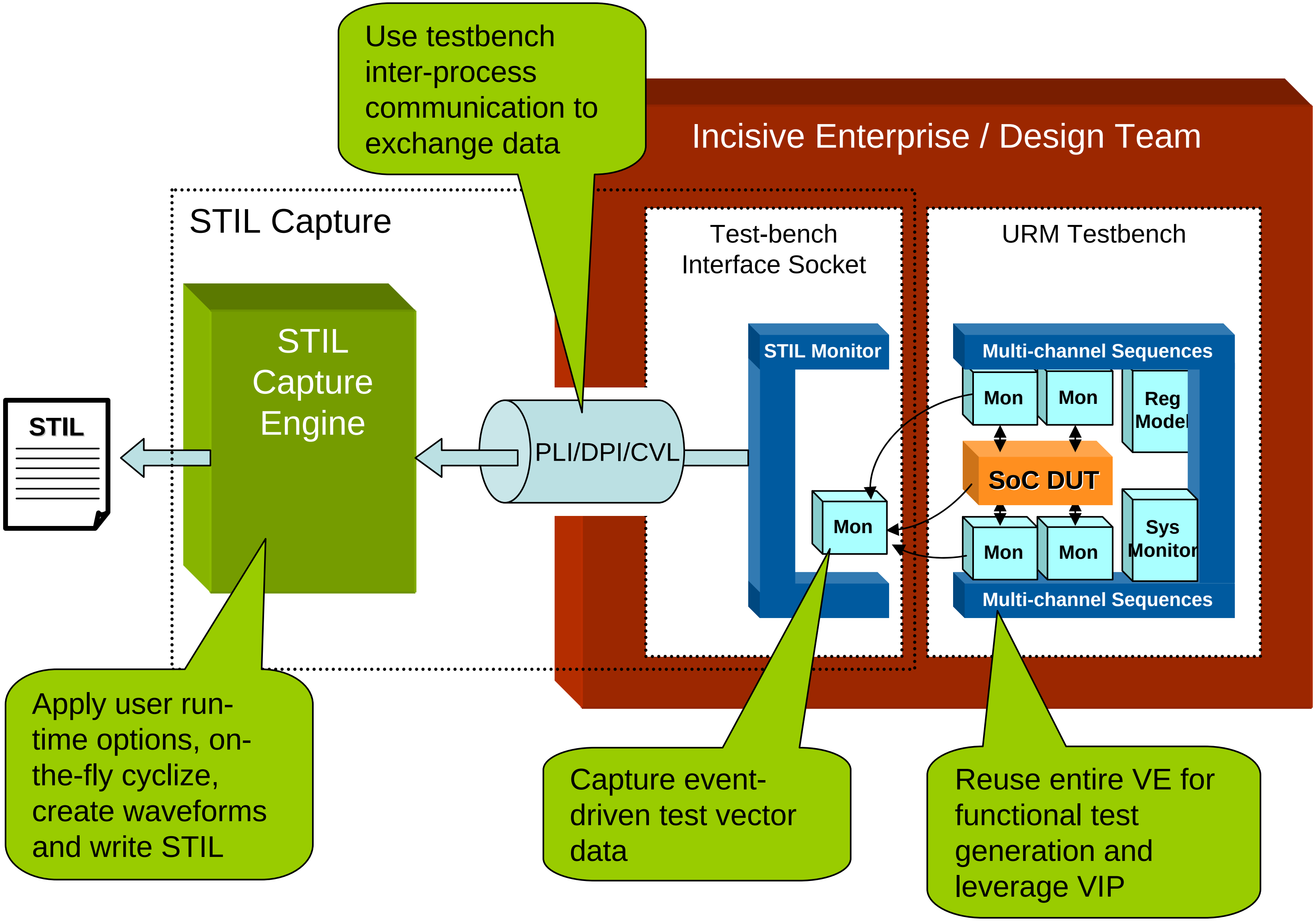
ATE Constraint: Waveform table limit == 9

Extending the current flow



- ✓ Metric-defined process
- ✓ Functional test scenario selection based on verification metrics
- ✓ Refinement based on test cost analysis
- ✗ No control over VCD
- ✗ Limited signal information in VCD (e.g. bidir signal handling)
- ✗ No feedback paths

Generating test programs



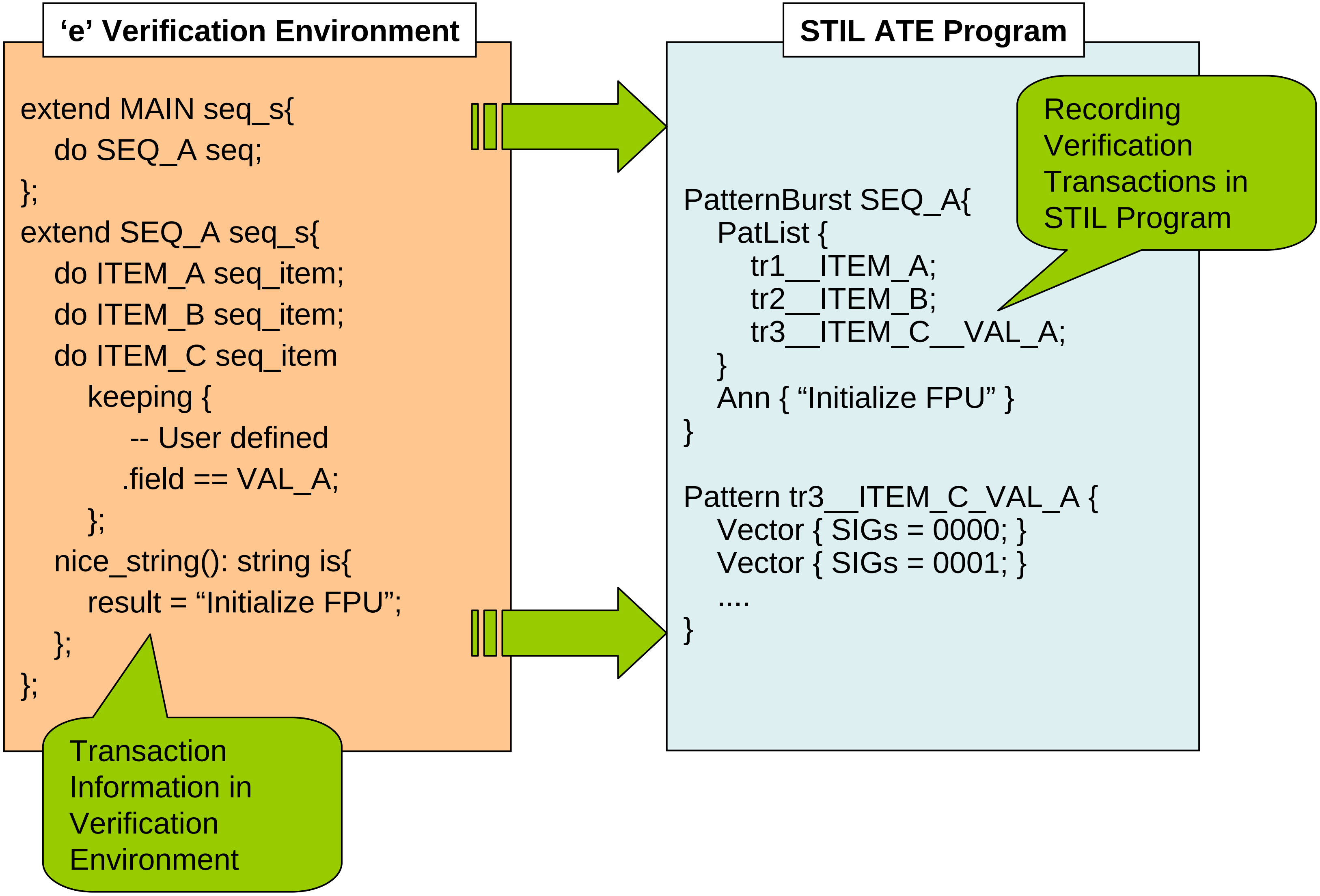
Use testbench inter-process communication to exchange data

Apply user run-time options, on-the-fly cyclize, create waveforms and write STIL

Capture event-driven test vector data

Reuse entire VE for functional test generation and leverage VIP

Transaction Recording



‘e’ Verification Environment

```
extend MAIN seq_s{
    do SEQ_A seq;
};
extend SEQ_A seq_s{
    do ITEM_A seq_item;
    do ITEM_B seq_item;
    do ITEM_C seq_item
        keeping {
            -- User defined
            .field == VAL_A;
        };
    nice_string(): string is{
        result = “Initialize FPU”;
    };
};
```

Transaction Information in Verification Environment

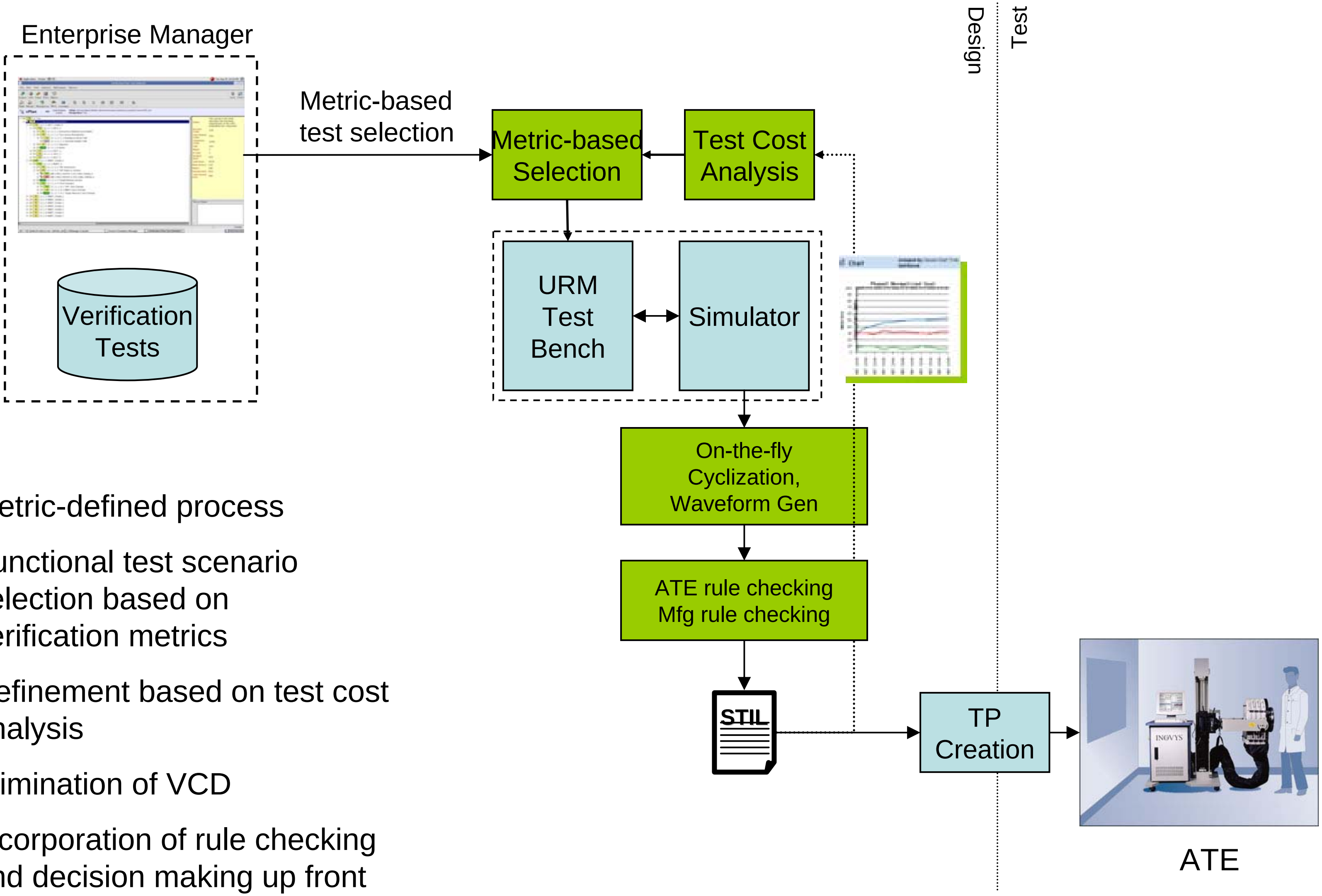
STIL ATE Program

```
PatternBurst SEQ_A{
    PatList {
        tr1__ITEM_A;
        tr2__ITEM_B;
        tr3__ITEM_C__VAL_A;
    }
    Ann { “Initialize FPU” }
}

Pattern tr3__ITEM_C_VAL_A {
    Vector { SIGs = 0000; }
    Vector { SIGs = 0001; }
    ....
}
```

Recording Verification Transactions in STIL Program

Extending the current flow



Enterprise Manager

Verification Tests

Metric-based test selection

Metric-based Selection

Test Cost Analysis

URM Test Bench

Simulator

On-the-fly Cyclization, Waveform Gen

ATE rule checking
Mfg rule checking

STIL

TP Creation

ATE

Design

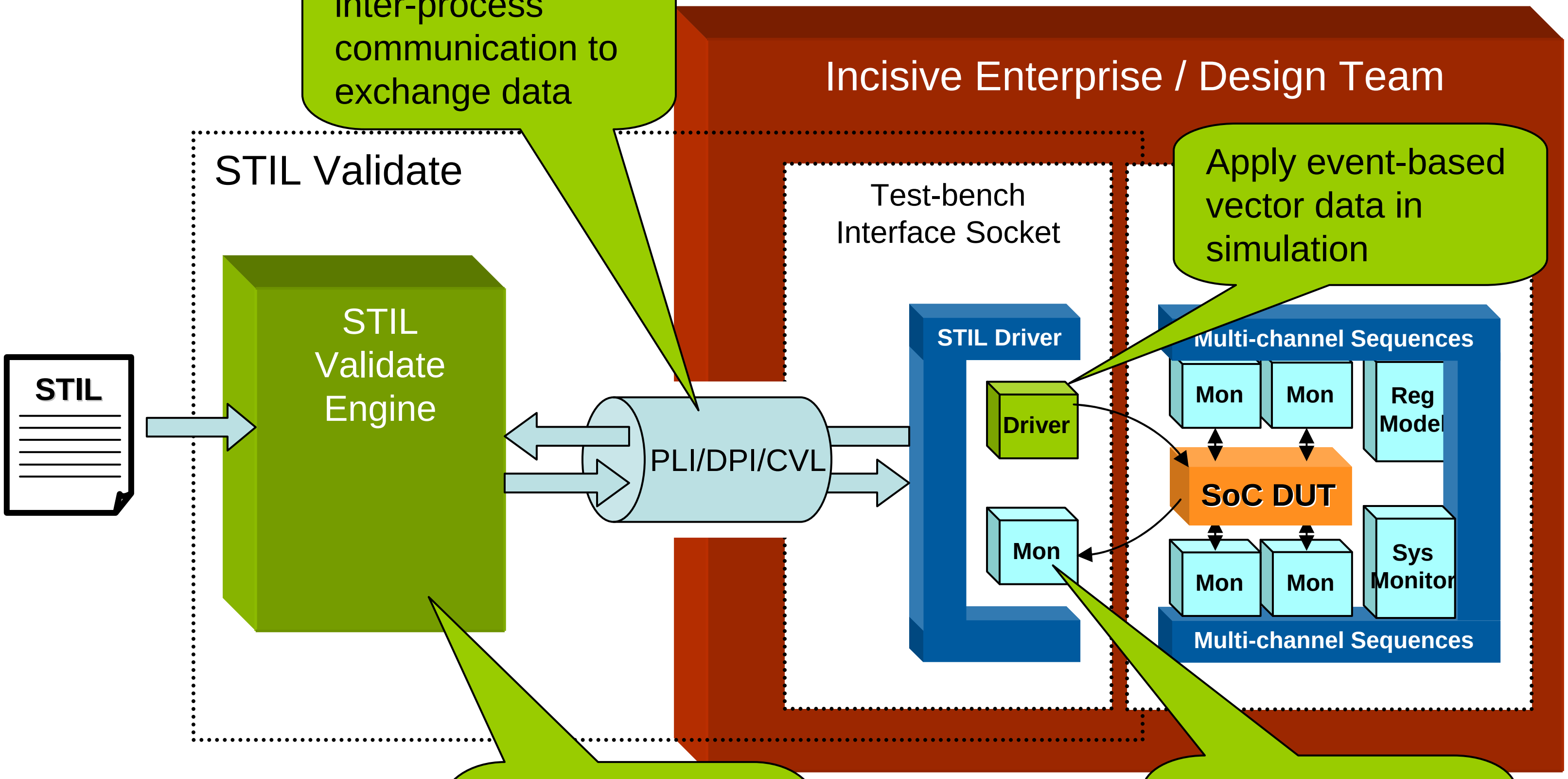
Test

- ✓ Metric-defined process
- ✓ Functional test scenario selection based on verification metrics
- ✓ Refinement based on test cost analysis
- ✓ Elimination of VCD
- ✓ Incorporation of rule checking and decision making up front
- ✗ No feedback paths

Feeding test data to design

- Debugging post-silicon design problems
  - ATE will say:
    - Failure at vector # 1249210
  - What was the device trying to accomplish at the time?
  - What was the state of the device at the time?
    - Simulation model
    - Real silicon
- Need a way to incorporate test programs into simulation
  - Simulate test programs in URM testbenches for checking and functional coverage collection
  - Debug test programs
  - Compare simulation data to real silicon data
- Need a way to incorporate metrics into test programs
  - Attributes that track progress
  - Can be connected with planning and measurement/reporting

Simulating test programs



Use testbench inter-process communication to exchange data

Apply event-based vector data in simulation

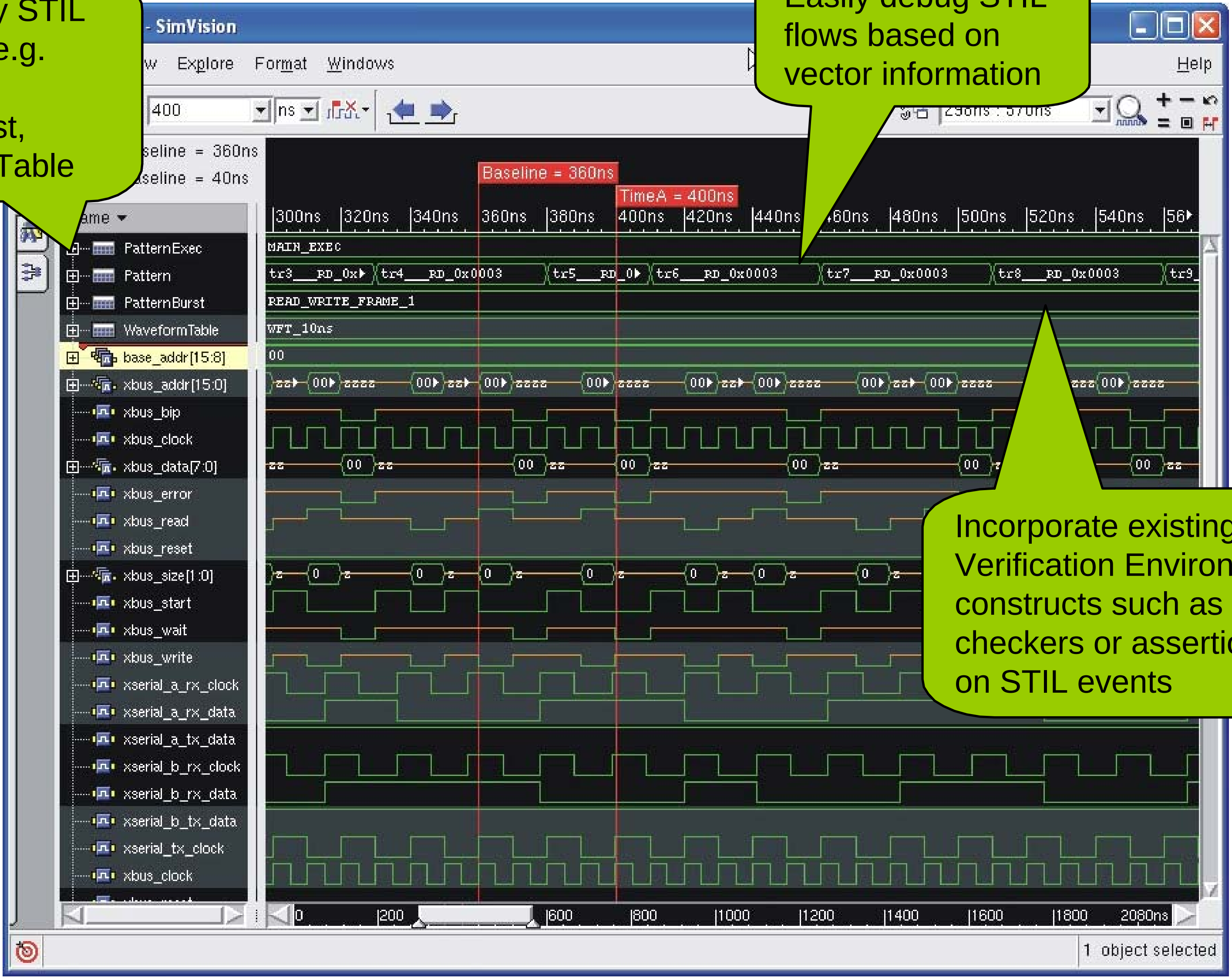
Parse & validate STIL test program, on-the-fly de-cyclize and send data to TIS

Execute STIL COMPARE statements based on DUT responses

Debugging test programs

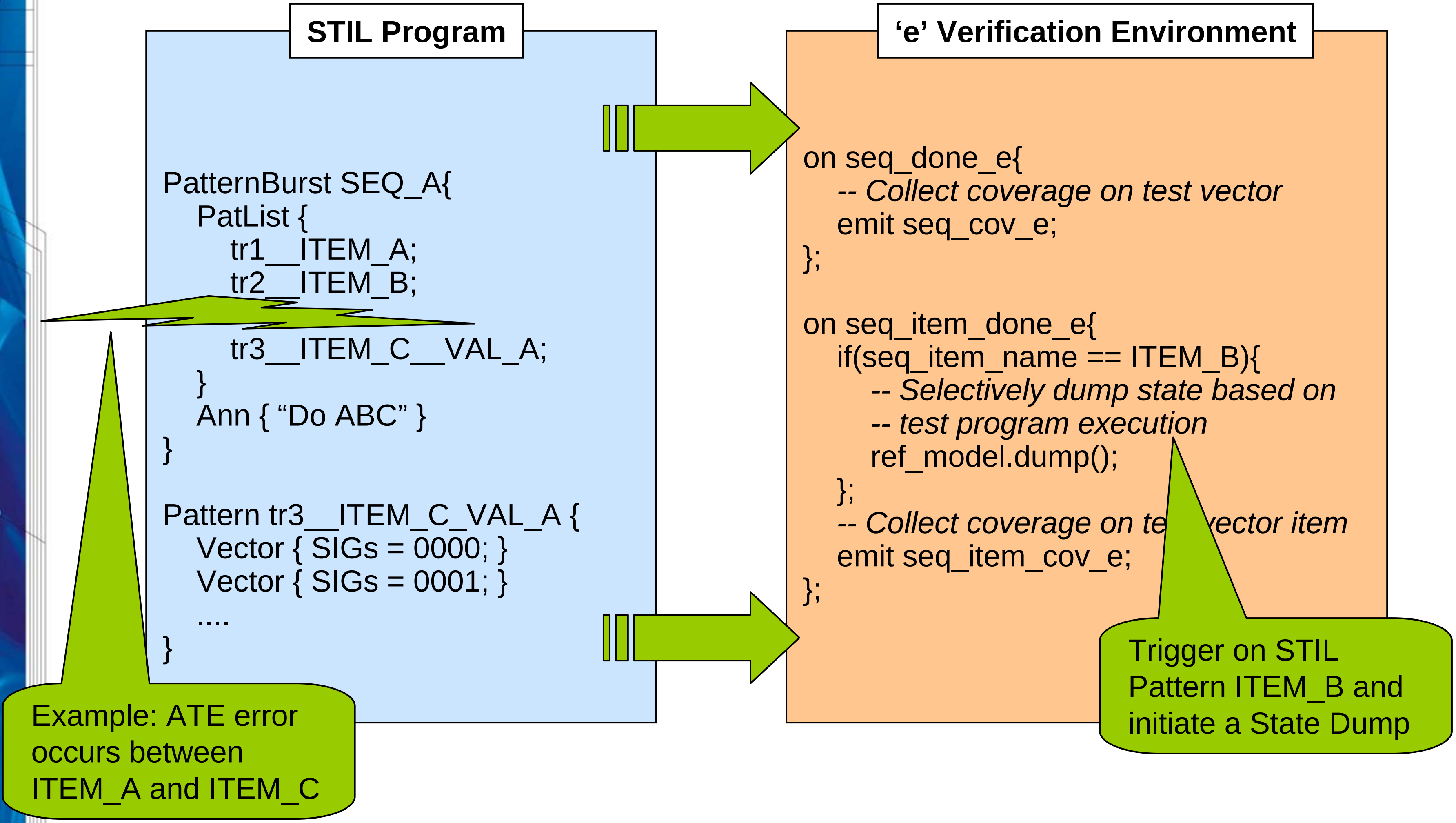
Display Key STIL Attributes, e.g. Pattern, PatternBurst, Waveform Table

Easily debug STIL flows based on vector information



Incorporate existing Verification Environment constructs such as checkers or assertions on STIL events

Test Program Assertions



STIL Program

```
PatternBurst SEQ_A{
    PatList {
        tr1__ITEM_A;
        tr2__ITEM_B;
        tr3__ITEM_C__VAL_A;
    }
    Ann { “Do ABC” }
}

Pattern tr3__ITEM_C_VAL_A {
    Vector { SIGs = 0000; }
    Vector { SIGs = 0001; }
    ....
}
```

‘e’ Verification Environment

```
on seq_done_e{
    -- Collect coverage on test vector
    emit seq_cov_e;
};

on seq_item_done_e{
    if(seq_item_name == ITEM_B){
        -- Selectively dump state based on
        -- test program execution
        ref_model.dump();
    };
    -- Collect coverage on te     vector item
    emit seq_item_cov_e;
};
```

Example: ATE error occurs between ITEM_A and ITEM_C

Trigger on STIL Pattern ITEM_B and initiate a State Dump

Plan & Manage validation

- Back to validation metrics
  - Functional and physical
- Plan to fulfilling metrics
  - Early assessment of which design attributes need to be validated in silicon
  - Build plan for complete metric-driven post-silicon validation
- Validation test suite
  - Test reuse with updated constraints
  - New tests that target heterogeneous metrics
    - Fault model coverage
    - Physical (e.g. CPF power targets)
- Back-annotation of plan with actual results from ATE and system
  - Measurements from validation process and experimentation
  - Blueprint for plan reuse project-to-project

Silicon Validation Plan

1 Silicon validation

1.1 Functional

1.1.1 Instruction Functionality

```
Code: coverage: STIL_monitor.instruction_cov_e (agent_name==*);
```

1.1.2 I/O

1.1.2.1 PCI Express Interface

```
Code: coverage: STIL_monitor.io_cov.PCIe_cov.lane_config
Code: coverage: STIL_monitor.io_cov.PCIe_cov.speed_config
```

1.2 Structural

...

1.3 Electrical

...

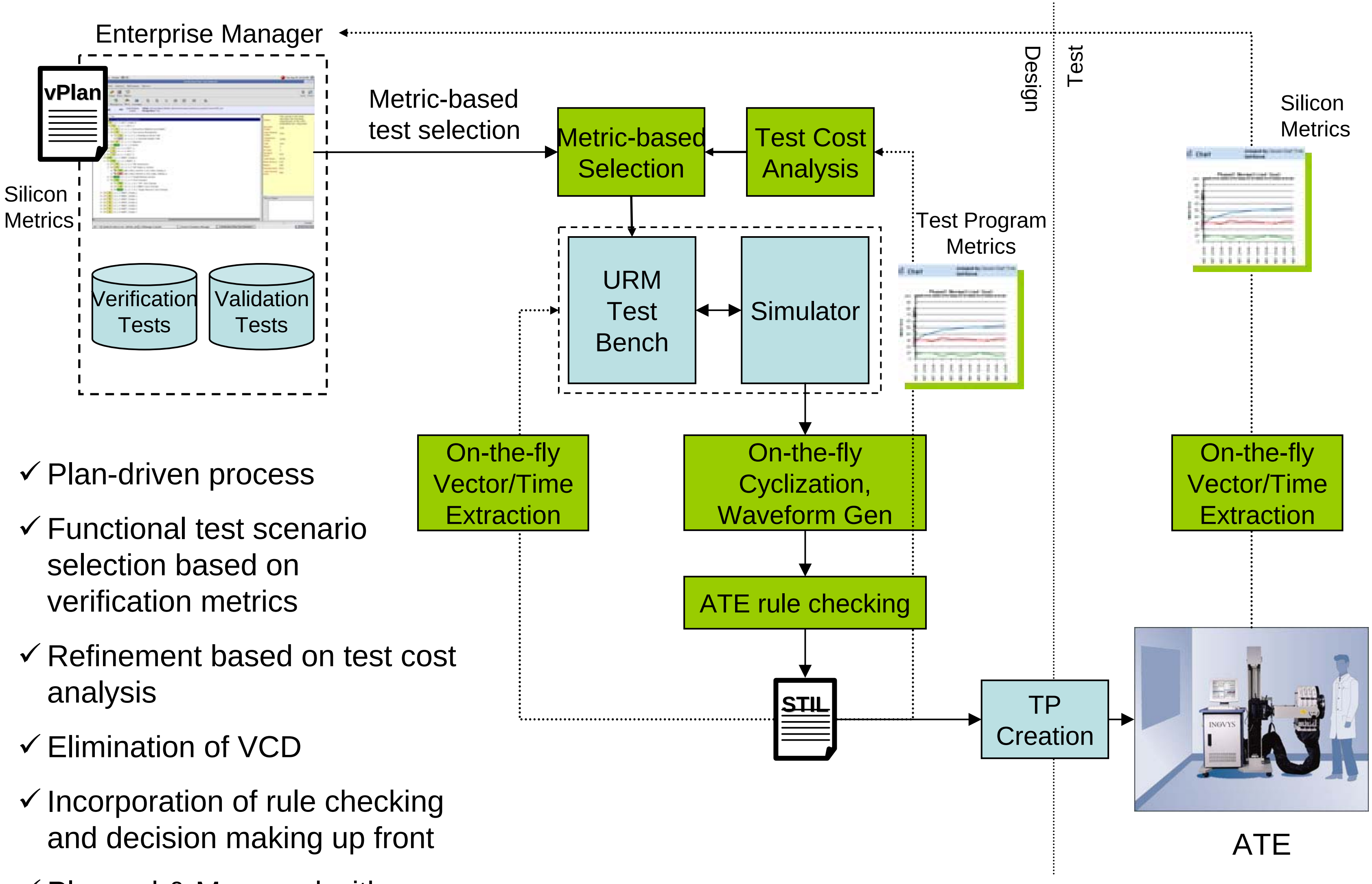
1.4 Power

...

1.5 ATE Environment

```
Code: coverage: ATE_monitor.voltage_margin
Code: coverage: ATE_monitor.temp_margin
```

Post-silicon validation flow



Enterprise Manager

vPlan

Silicon Metrics

Verification Tests

Validation Tests

Metric-based test selection

Metric-based Selection

Test Cost Analysis

URM Test Bench

Simulator

Test Program Metrics

Design

Test

Silicon Metrics

On-the-fly Vector/Time Extraction

On-the-fly Cyclization, Waveform Gen

On-the-fly Vector/Time Extraction

ATE rule checking

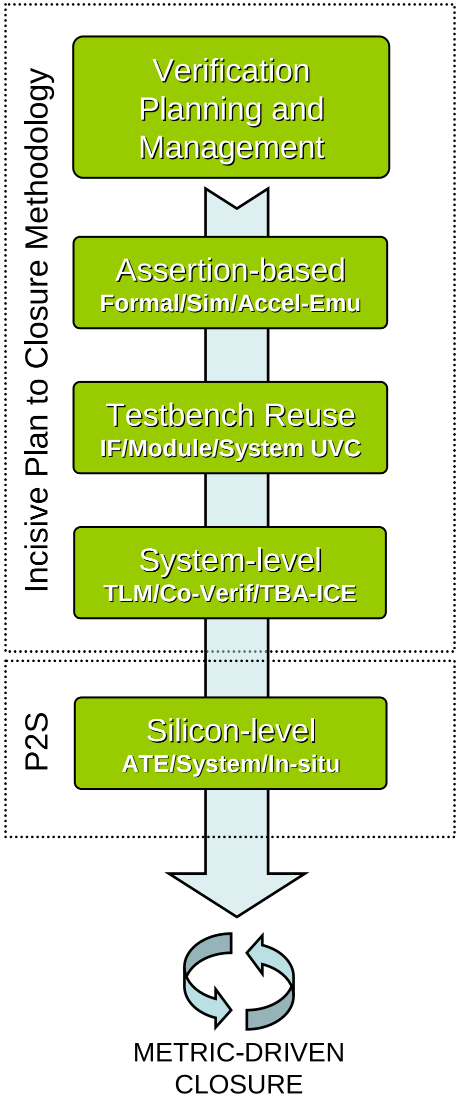
STIL

TP Creation

ATE

- ✓ Plan-driven process
- ✓ Functional test scenario selection based on verification metrics
- ✓ Refinement based on test cost analysis
- ✓ Elimination of VCD
- ✓ Incorporation of rule checking and decision making up front
- ✓ Planned & Managed with feedback paths

Plan to Silicon (P2S)



Verification Planning and Management

Assertion-based
Formal/Sim/Accel-Emu

Testbench Reuse
IF/Module/System UVC

System-level
TLM/Co-Verif/TBA-ICE

Silicon-level
ATE/System/In-situ

METRIC-DRIVEN CLOSURE

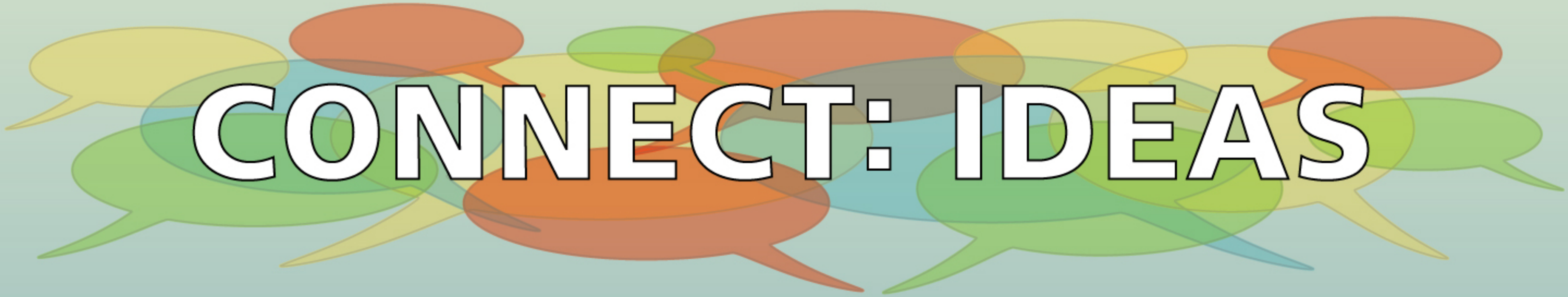
Benefits of P2S

- Enhanced quality of result
  - Plan, do not react
  - Anticipate and learn
- Increased design productivity
  - Improved specialist integration
  - Verification, test, diagnostics, yield
- Reduced support overhead
  - Less friction between design for test
  - Less black magic
- Significant cost-of-test savings
  - Optimized functional test content targeted at different parts of the post-silicon life-cycle

cādence™

cadence designer network

cdn LIVE™



CONNECT: IDEAS

CDNLive! 2007 Silicon Valley